MERGING, REPAIRING AND QUERYING INCONSISTENT DATABASES WITH FUNCTIONAL AND INCLUSION DEPENDENCIES

Luciano Caroprese, Sergio Greco and Ester Zumpano
DEIS, Università della Calabria
87030 Rende, Italy

Keywords: Data Integration, Inconsistent Databases, Consistent Query Answers.

Abstract: In this paper a framework for merging, repairing and querying inconsistent databases is presented. The framework, considers integrity constraints defining primary keys, foreign keys and general functional dependencies. The approach consists of three steps: i) merge of the source databases by means of integration operators or general SQL queries, to reduce the set of tuples coming from the source databases which are inconsistent with respect to the constraints defined by the primary keys, ii) repair of the integrated database by completing and/or cleaning the set of tuples which are inconsistent with respect to the inclusion dependencies (e.g. foreign keys), and iii) compute consistent answers over repaired databases which could be still inconsistent with respect to the functional dependencies. The complexity of merging, repairing and computing consistent answers will be show to be polynomial and a prototype of a system integrating databases and computing queries over possible inconsistent databases will be presented.

1 INTRODUCTION

The primary task of information integration consists in providing a uniform integrated access to multiple independent information sources, which are both distributed and potentially heterogeneous and whose contents are strictly related. However the integrated view, constructed by integrating the information provided by the different data sources, could potentially contain inconsistent data, i.e. it can violate some of the constraints defined on the data.

Example 1 Consider the database consisting of the relation *Employee(Name, Age, Salary)* where the attribute *Name* is a key for the relation. Assume there are two different instances for the relations *Employee*:
$\{Employee(Mary, 28, 20), Employee(Peter, 47, 50)\}$
$\{Employee(Mary, 31, 30), Employee(Peter, 47, 50)\}$

If the integration of the two relations $Employee_1$ and $Employee_2$ is performed by using the union operator (i.e. by considering the union of the tuples in both the relations) the following integrated relation is obtained: $\{Employee(Mary, 28, 20),\ Employee(Mary, 31, 30), Employee(Peter, 47, 50)\}$ which does not anymore satisfy the key constraint.

As showed in the previous example, in some case the integrated database may be not consistent with respect to the integrity constraints. In such a case there are two possible solutions which have been investigated in the literature (Agarwal et al., 1995; Arenas et al., 1999; Bry, 1997; Dung, 1996; Greco et al., 2001; Greco and Zumpano, 2000; Lin, 1996; Lin, 1996; Lin and Mendelzon, 1999): repairing the database by inserting or deleting tuples, so that the resulting database is consistent, or computing consistent answers over the inconsistent database. Intuitively, a repair of the database consists in deleting or inserting a minimal number of tuples so that the resulting relation is consistent. The second approach, considering the computation of consistent answer, is based on the fact that, since the data are not consistent, answers may be certain or uncertain.

Example 2 Consider again the integrated database $Employee$ reported in Example 1. In this case there are two possible repaired databases each obtained by deleting one of the two tuples whose value of the attribute $Name$ is $Mary$. As regards the answer to the query asking for *the age of* $Peter$ this is constituted by the certain set $\{47\}$, whereas the answer to the query asking for *the age of* $Mary$ produces the following set of uncertain values $\{28, 31\}$. □

Thus, in this paper we also consider, other than the merging of databases, the repairing of databases and the computation of consistent answers. In our framework we consider integrity constraints defining pri-

Caroprese L., Greco S. and Zumpano E. (2006).
MERGING, REPAIRING AND QUERYING INCONSISTENT DATABASES WITH FUNCTIONAL AND INCLUSION DEPENDENCIES.
In *Proceedings of the Eighth International Conference on Enterprise Information Systems - DISI*, pages 38-45
DOI: 10.5220/0002458900380045
Copyright © SciTePress

mary keys, functional dependencies and foreign keys. As above explained, the motivation to consider general functional dependencies, and special forms of functional dependencies such as primary keys, is that primary keys are used in the merging phase, whereas functional dependencies are used to compute consistent answers. Foreign keys are used to repair the database so that the resulting database satisfy them. The process can be performed by (i) *completing* the database, i.e. by inserting tuples, and/or (ii) by *cleaning* the database, i.e. by removing tuples.

The proposed framework for merging, repairing and querying inconsistent databases with functional and inclusion dependencies, has been implemented in a system prototype, developed at the University of Calabria.

2 DATABASE INTEGRATION AND QUERYING

Integrating and querying data from different sources consists of three main steps: the first in which the various relations are merged together to reduce the set of inconsistent tuples; the second in which some tuples are removed or inserted into the database in order to satisfy some of the *integrity* constraints; the third in which the query answer, consisting of the set of *certain* and *uncertain* tuples, is computed. Before formally introducing the problems related to the merging, repairing and querying of inconsistent databases let us introduce some basic definitions and notations. Generally, a database D has associated a schema $\mathcal{DS} = \langle Rs, \mathcal{IC} \rangle$ which defines the intentional properties of D: Rs denotes the structure of the relations and $\mathcal{IC}$ contains the set of integrity constraints. In this paper we concentrate on integrity constraints consisting of functional dependencies, including constraints defining primary keys, and inclusion dependencies expressing foreign keys. Thus, our set of integrity constraints $\mathcal{IC}$ can be partitioned into two sets: $\mathcal{FD}$, consisting of functional dependencies, and $\mathcal{FK}$, consisting of foreign keys, i.e. $\mathcal{IC} = \mathcal{FD} \cup \mathcal{FK}$. Moreover, we denote with $\mathcal{PK}$ the subset of the functional dependencies defining primary keys.

Given a relation R, a *functional dependency* $X \rightarrow Y$ over R can be expressed by a formula of the form

$\forall(X,Y,Z,U,V)[\,R(X,Y,U) \ \wedge\ R(X,Z,V) \quad \supset Y = Z\,]$

where X, Y, Z, U, V are lists of variables and Y, Z may be empty lists.

Let D be a database, R a relation of D and t a tuple in R, then we denote by: i) *attr(R)* the set of attributes of R, ii) *pkey(R)* the set of attributes in the primary key of R, iii) $fd(R)$ the set of functional dependencies of R, iv) $pkey(t)$ the values of the key attributes of t.

Databases Merging. The database merging problem consists in the merging of k databases $D_1 = \{R_{1,1}, ...R_{1,n_1}\}$, ..., $D_k = \{R_{k,1}, ...R_{k,n_k}\}$. For the sake of simplicity we assume that all databases contain the same number of relations (i.e. $n_1 = \cdots = n_k = n$) and that for every $1 \leq j \leq k$, $R_{1,j}, ..., R_{k,j}$ refer to the same concept. The integrated database D consists of n relations $T_1, ..., T_n$ where every T_j is obtained by merging the relations $R_{1,j}, ..., R_{k,j}$. Thus, the database integration problem consists in the integration of a set of relations $R_{1,j}, ..., R_{k,j}$ into a relation T_j by means of a (binary) integration operator $\diamond$, i.e. computes $T_j = (...(R_{1,j} \diamond R_{2,j}) \diamond \cdots) \diamond R_{k,j}$. We assume that relations associated with the same class of objects have been *homogenized* with respect to a common ontology, i.e. attributes denoting the same concepts have the same name (Yan and Ozsu, 1999). We say that two homogenized relations R and S, associated with the same concept, are *overlapping* if $pkey(R) = pkey(S)$. In the following we assume that relations associated with the same class of objects are overlapping.

Several merge operators have been proposed in the literature. We recall here the Match Join operator (Yan and Ozsu, 1999), the Merging by Majority operator (Lin and Mendelzon, 1999), the Merge operator (Greco et al., 2001), the Prioritized Merge operator (Greco et al., 2001). Before presenting, in an informal way, these operators we introduce some preliminary definition and formally define desirable properties of integration operators.

Definition 1 Given two relations R and S such that $attr(R) \subseteq attr(S)$ and two tuples $t_1 \in R$ and $t_2 \in S$, we say that t_1 is *less informative* than t_2 ($t_1 \ll t_2$) if for each attribute A in $attr(R)$, $t_1[A] = t_2[A]$ or $t_1[A] =\perp$, where $\perp$ denotes the null value. Moreover, given two relations R and S, we say that $R \ll S$ if $\forall t_1 \in R, \exists t_2 \in S$ s.t. $t_1 \ll t_2$. □

Definition 2 Let R and S be two relations, a binary operator $\diamond$ such that: (i) $attr(R \diamond S) = attr(R) \cup attr(S)$, (ii) $R \bowtie S \ll R \diamond S$, (iii) $(R \diamond R) = R$ (*idempotency*). is called *integration* or *merge operator*. Moreover, an integration operator $\diamond$ is said to be (i) *commutative*, if $R \diamond S = S \diamond R$. (ii) *associative*, if $(R \diamond S) \diamond T = R \diamond (S \diamond T)$. (iii) *complete* (or *lossless*), if for all R and S, $R \ll (R \diamond S)$ and $S \ll (R \diamond S)$; (iv) *dependency preserving*, if for all R and S, is $(R \diamond S) \models (fd(R) \cap fd(S))$; (v) *correct* if $\forall t \in R \diamond S\ \exists t'$ s.t. ($t' \in R$ or $t' \in S$) and $t' \ll t$ □

Informally, if an integration operator is both correct and complete it preserves the information provided by the sources. In fact, it could modify some input tuples by replacing null values with not null ones, but all the associations of not null values which were contained in the source relations will be inserted into the result (completeness) and no association of not null values which was not contained in the source relations will be inserted into the result (correctness). Moreover, note that integrating more than two relations by means of a not associative integration operator, may give different results, if the integration operator is applied in different orders. Thus, even if the associative property is desirable, it is not satisfied by several operators defined in the literature.

Repairing inconsistent databases. Let us first introduce the formal definition of consistent database and repairs.

Definition 3 Given a database schema $\mathcal{DS} = \langle Rs, \mathcal{IC} \rangle$ and a database instance D over Rs, we say that D is *consistent* if $D \models \mathcal{IC}$, i.e. if all integrity constraints in $\mathcal{IC}$ are satisfied by D, otherwise it is *inconsistent*. $\Box$

Example 3 The database of Example 1, derived from the union of the two source databases, is inconsistent as the functional dependency

$$\forall (K, A_1, S_1, A_2, S_2)[\, Employee(K, A_1, S_1)\ , \\ Employee(K, A_2, S_2) \supset A_1 = A_2, S_1 = S_2\,]$$

stating that K is a key for the relation $Employee$ is not satisfied. $\Box$

Informally, a repair R for a (possibly inconsistent) database D is a minimal, consistent set of insert and delete operations (denoted, respectively, by R^+ and R^-) which makes D consistent. Given a repair R, R^+ denotes the set of tuples which will be added to the database whereas R^- denotes the set of tuples of D which will be deleted.

Given two pairs of sets of atoms $S = (S^+, S^-)$ and $R = (R^+, R^-)$, we say that $S \subseteq R$ if both $S^+ \subseteq R^+$ and $S^- \subseteq R^-$. We also say that $S \subset R$ if $S \neq R$ and $S \subseteq R$.

Definition 4 *(Arenas et al., 1999)* Let $\mathcal{DS} = \langle Rs, \mathcal{IC} \rangle$ be a database schema and D be a (possibly inconsistent) database over Rs, then a *repair* for D is a pair of sets of atoms (R^+, R^-) such that: (i) $R^+ \cap R^- = \emptyset$; (ii) $D \cup R^+ - R^- \models \mathcal{IC}$ and (iii) there is no repair $(S^+, S^-) \subset (R^+, R^-)$.

The database $R(D) = D \cup R^+ - R^-$, obtained from the application of R to D, will be called the *repaired database*. $\Box$

Thus, repaired databases are consistent databases which are derived from the source database by means of a minimal set of insertion and deletion of tuples.

Example 4 Assume we are given the database $D = \{p(a), p(b), q(a),\ q(c)\}$, with the *inclusion dependency*: $(\forall\, X)\ [\ p(X)\ \supset\ q(X)\ ]$. D is inconsistent since $p(b) \supset q(b)$ is not satisfied. The repairs for D are $R_1 = (\{q(b)\}, \emptyset)$ and $R_2 = (\emptyset, \{p(b)\})$ producing, respectively, the repaired databases $R_1(D) = \{p(a), p(b), q(a), q(c), q(b)\}$ and $R_2(D) = \{p(a), q(a), q(c)\}$. $\Box$

Queries over inconsistent databases. A (relational) query over a database defines a function from the database to a relation. It can be expressed by means of alternative equivalent languages such as relational algebra, 'safe' relational calculus or 'safe' non-recursive Datalog (Abiteboul and al., 1994; Ullman, 1998) (i.e., safe Datalog without disjunction, classical negation and recursion). Thus, a query is a pair $(g, \mathcal{P})$, where $\mathcal{P}$ is an expression of the query language and g is a predicate symbol specifying the output (derived) relation.

Definition 5 Given a database schema $\mathcal{DS} = \langle Rs, \mathcal{IC} \rangle$ and a database D over Rs, an atom A is *true* (resp. *false*) with respect to $(D, \mathcal{IC})$ if A belongs to all repaired databases (resp. there is no repaired database containing A). The set of atoms which are neither true nor false are *undefined*. $\Box$

Thus, true atoms appear in all repaired databases (Arenas et al., 1999), whereas undefined atoms appear in a proper subset of repaired databases.

Definition 6 Given a database schema $\mathcal{DS} = \langle Rs, \mathcal{IC} \rangle$ and a database D over Rs, the application of $\mathcal{IC}$ to D, denoted by $\mathcal{IC}(D)$, defines three distinct sets of atoms: the set of *true atoms* $\mathcal{IC}(D)^+$, the set of *undefined atoms* $\mathcal{IC}(D)^u$ and the set of *false atoms* $\mathcal{IC}(D)^-$. $\Box$

Definition 7 Given a database schema $\mathcal{DS} = \langle Rs, \mathcal{IC} \rangle$, a database D over Rs and a query $Q = (g, \mathcal{P})$, the *consistent answer* of the query Q on the database D, denoted as $Q(D, \mathcal{IC})$, gives three sets, denoted $Q(D, \mathcal{IC})^+$, $Q(D, \mathcal{IC})^-$ and $Q(D, \mathcal{IC})^u$, containing, respectively, the sets of g-tuples which are $true$ (i.e. belonging to $Q(D')$ for all repaired databases D'), *false* (i.e. not belonging to $Q(D')$ for all repaired databases D') and *undefined* (i.e. set of tuples which are neither true nor false). $\Box$

3 DATABASE INTEGRATION

The Match Join Operator. The *Match Join* operator, $\bowtie$, proposed in (Yan and Ozsu, 1999), manufactures tuples in the integrated relation by performing the outer-join of the $ValSet$ of each attribute, where the $ValSet$ of an attribute A is the union of the projections of each overlapping relations on $\{K, A\}$.

Example 5 Consider the following two overlapping relations S_1 and S_2:

K	$Title$	$Author$
1	$Moon$	$Greg$
2	$Money$	$Jones$
3	Sky	$Jones$

S_1

K	$Title$	$Author$	$Year$
3	$Flowers$	$Smith$	1965
4	Sea	$Taylor$	1971
7	Sun	$Steven$	1980

S_2

and suppose the attribute $Author$ is functionally dependent on the attribute $Title$, i.e. $Title \rightarrow Author$.

The relation obtained T by applying the Match Join operator to the relations S_1 and S_2, i.e. $T = S_1 \bowtie S_2$, is the following:

K	$Title$	$Author$	$Year$
1	$Moon$	$Greg$	$\perp$
2	$Money$	$Jones$	$\perp$
3	Sky	$Jones$	1965
3	Sky	$Smith$	1965
3	$Flowers$	$Smith$	1965
3	$Flowers$	$Jones$	1965
4	Sea	$Taylor$	1971
7	Sun	$Steven$	1980

T

where $\perp$ denotes the null value.

The Match Join operator is complete, but it is not correct, since it mixes values coming from different tuples with the same key in all possible ways. As a consequence, when applying the Match Join operator to S_1 and S_2 we obtain an integrated view T violating the functional dependency $Title \rightarrow Author$. Thus the Match Join operator produces tuples containing associations of values that may be not present in any original relation and the integration process may generate a relation which is not anymore consistent w.r.t. the functional dependency.

The Merging by Majority Operator. The *Merging by Majority* operator, proposed in (Lin and Mendelzon, 1999) tries to remove conflicts taking into account the majority view of the knowledge bases, i.e. it maintains the (not null) value which is present in the majority of the knowledge bases. Thus the operator constructs an integrated relation containing *generalized* tuples, i.e. tuples where each attribute value is a simple value, if the information respects the majority criteria, or a set, if the technique doesn't resolve the conflict.

Example 6 Consider the database consisting of the relation $Bib(Author, Title, Year)$ which collect information regarding author, title and year of publication of papers. Assume to have the following three database instances:

$$\{Bib(John, T_1, 1980), Bib(Mary, T_2, 1990)\},$$
$$\{Bib(John, T_1, 1981), Bib(Mary, T_2, 1990)\},$$
$$\{Bib(John, T_1, 1980), Bib(Frank, T_3, 1990)\}.$$

From the integration of the above three databases we obtain the database Bib. $\{Bib(John,\ T_1,\ 1980),\ Bib(Mary,\ T_2,\ 1990),\ Bib(Frank, T_3, 1990)\}$.

Note that, the Merging by Majority operator removes the conflict about the year of publication of the paper T_1 written by the author John observing that two of the three source relations, that have to be integrated, store the value 1980; thus the information that is maintained is the one which is present in the majority of the knowledge bases.

However, the Merging by Majority technique does not resolve conflicts in all cases since information is not always present in the majority of the databases and, therefore, it is not always possible to choose among alternative values. In this case the integrated database contains generalized tuples.

For instance, the merging of the two relation S_1 and S_2 of Example 6 gives the "nested" relation $\{Bib(John, T_1, \{1980,\ 1981\}),\ Bib(Mary,\ T_2,\ 1990),\ Bib(Frank, T_3, 1990)\}$.

Here the first tuple states that the year of publication of the book written by John with title T_1 can be one of the values belonging to the set {1980, 1981}.

Thus the Merging by Majority approach could fail when, for a field, the database majority does not agree on a value. In this case, for each of this field, a set of possible values is associated in the integrated databases. The Merging by Majority operator is correct, but it is not complete.

The Merge Operator. Given two overlapping relations S_1 and S_2, the Merge operator, introduced in (Greco et al., 2001), applied to S_1 and S_2, $S_1 \boxtimes S_2$, produces a relation S obtained by completing the information coming from each input relation with that coming from the other one. More specifically, it computes the full outer join and extends tuples coming from S_1 (resp. S_2) with the values of tuples of S_2 (resp. S_1) having the same key.

Example 7 Consider the relations $S1$ and $S2$ reported in the Example 5. The relation T, obtained by merging S_1 and S_2 through the *merge operator*, i.e. $T = S_1 \boxtimes S_2$, is the following:

K	$Title$	$Author$	$Year$
1	$Moon$	$Greg$	$\perp$
2	$Money$	$Jones$	$\perp$
3	Sky	$Jones$	1965
3	$Flowers$	$Smith$	1965
4	Sea	$Taylor$	1971
7	Sun	$Steven$	1980

T

Note that the integrated relation does not violate the functional dependency $Title \rightarrow Author$. $\Box$

The Merge operator is correct and complete. Moreover, on the contrary of the Match Join operator , which mixes values coming from different tuples with the same key in all possible ways, it only tries to derive unknown values so that number of integrated tuples violating the key constraint is reduced.

The Prioritized Merge Operator. The *Prioritized Merge operator*, $\lhd$, introduced in (Greco et al., 2001), is an asymmetric operator which, if conflicting tuples are detected, gives preference to data coming from the source on which the user expressed preference. More specifically, given two source relations, S_1 and S_2, the prioritized merge operator applied to S_1 and S_2, $S = S_1 \lhd S_2$, produces a relation S which includes all tuples of the left relation and only the tuples of the right relation whose key does not identify any tuple in the left relation. Moreover, only tuples 'coming' from the left relation are extended since tuples coming from the right relation, joining some tuples coming from the left relation, are not included. Thus, when integrating relations conflicting on the key attributes, the prioritized merge operator gives preference to the tuples of the left side relation and completes them with values taken from the right side relation.

Example 8 Consider the source relations S_1 and S_2 of Example 5. The relation $T = S_1 \lhd S_2$ is:

K	$Title$	$Author$	$Year$
1	$Moon$	$Greg$	$\perp$
2	$Money$	$Jones$	$\perp$
3	Sky	$Jones$	1965
4	Sea	$Taylor$	1971
7	Sun	$Steven$	1980

T

The merged relation obtained in this case differs from the one of Example 7 because it does not contain the tuple $(3, Flowers, Smith, 1965)$ coming from the right relation S_2. $\Box$

Obviously, the prioritized merge operator is correct, but it is not complete as in the presence of conflicting tuples it holds the values coming from the left relation. Moreover, given two overlapping relations S_1 and S_2, then (i) $S_1 \lhd S_2 \subseteq S_1 \boxtimes S_2$, (ii) $S_1 \boxtimes S_2 = (S_1 \lhd S_2) \cup (S_2 \lhd S_1)$ and (iii) $S_1 \lhd S_1 = S_1$.

More General Techniques. In some cases the use of predefined operators does not satisfy the intended integration strategy. Thus, we present a more general merging strategy allowing to specify for each attribute A the integration task to be performed. Given a set of homogenized relations $S_1, ..., S_n$, a *reconciled* relation S is s.t.: i) $attr(S) = \bigcup_{i=1}^{n} attr(S_i)$, ii) it contains all tuples $t \in S_i$, $1 \le i \le n$, completed with $\perp$ for all attributes belonging to $attr(S) - attr(S_i)$. Thus, starting from the reconciled relation S, we consider a more general strategy based on i) the collection of tuples with the same value for the key attributes into a 'nested' tuple where each non key attribute, say A, contains the list $[a_1, ..., a_n]$ of the values of the attribute A present in the tuples of S which have to be merged and ii) the application of a polynomial function f_i to $[a_1, ..., a_n]$. The specialization of the functions f_i permits us to express most of the integration operators defined in literature. For instance, the merging by majority technique is obtained by specializing all f_i to select the element which occurs a maximum number of times in the list. In other cases the function f_i computes a value such as the maximum, minimum, average, etc.

Fact 1 *The complexity of constructing the merged database by means of an integration operators is polynomial time.* $\Box$

4 COMPUTING DATABASE REPAIRS

In this section we consider the computation of repairs with respect to a set of inclusion dependencies.

An *inclusion dependency* is an integrity constraint of the form $\forall(X, Y)[p(X, Y) \supset \exists Z q(X, Z)]$ whose meaning in that if there exists a tuple (x, y) in p, then there must exist a tuple (x, z) in q for some value z.

In order to repair a database not satisfying an inclusion dependency of the above form, two solutions can be performed. In more details, for every tuple (x, y) of p such that there is no tuple (x, z) (for all possible values z) in q i) delete the tuple (x, y) from p or insert a tuple $(x, \perp_1)$ into q where $\perp_1$ is a null value if the corresponding attribute is not a foreign key or a new value different from all values present in the database. In the first case we say that the database is *cleaned*, as we consider the inconsistent data as not correct, whereas, in the second case we say that the database is *completed*, as we consider the data which are inconsistent (with respect to inclusion dependencies) as correct.

Proposition 1 *Let D be a database, $Q = (g, \mathcal{P})$ a query and $\mathcal{IC}$ a set of inclusion dependencies. Then, (i) a repair for D always exists, (ii) the computation of a repair, selected nondeterministically, can be done in polynomial time.* $\Box$

The above proposition can be easily proved as a cleaning repair always exists and it can be computed by deleting tuples, one-at-time, non satisfying inclusion dependencies.

Theorem 2 *Let D be a database and $\mathcal{IC}$ a set of inclusion dependencies defining foreign keys. Then, (i) a completing repair for D always exists, (ii) the computation of a completing repair, can be done in polynomial time.* $\Box$

5 QUERYING INCONSISTENT DATABASES

Given a query $\mathcal{Q} = \langle g, \mathcal{P} \rangle$ and a consistent database D, the answer to the query Q applied to D is $\mathcal{Q}(D) = \{t | g(t) \in \mathcal{P}(D)\}$, returns the set of g-tuples belonging to the result obtained from the evaluation of $\mathcal{P}$ over D. Clearly, in this case the query $\mathcal{Q}(D)$ defines a function from the database to a relation.

The answer to a query $\mathcal{Q}$ over a database D which may be inconsistent with respect to a set of integrity constraints is $\mathcal{Q}(D, \mathcal{IC}) = \{\{t | g(t) \in \mathcal{P}(R(D))\} | \exists R \in \mathcal{R}(D)\}$, where $\mathcal{R}(D)$ is the set of repairs of D and $R(D)$ is the repaired database w.r.t R. Thus, in this case we may have more than one outcome, as we evaluate the query over all possible repaired databases. Therefore, in this case the $\mathcal{Q}(D, \mathcal{IC})$ defines a multi-valued function from the database to a set of relations.

A deterministic answers can be obtained by considering either the union or the intersection of the sets in $\mathcal{Q}(D, \mathcal{IC})$. More specifically, the certain answer consists of the tuples belonging to all sets in $\mathcal{Q}(D, \mathcal{IC})$, whereas the possible answer consists of the tuples belonging to some set in $\mathcal{Q}(D, \mathcal{IC})$. Therefore, the certain answer is $\mathcal{Q}(D, \mathcal{IC})_c = \bigcap_{M \in \mathcal{Q}(D, \mathcal{IC})} M$, whereas the possible answer is $\mathcal{Q}(D, \mathcal{IC})_p = \bigcup_{M \in \mathcal{Q}(D, \mathcal{IC})} M$. As a consequence, the uncertain answer is $\mathcal{Q}(D, \mathcal{IC})_u = \mathcal{Q}(D, \mathcal{IC})_p - \mathcal{Q}(D, \mathcal{IC})_c$.

For instance, in Example 4, the set of true tuples are those belonging to the intersection of the two models, that is $p(a)$, $q(a)$ and $q(c)$, whereas the set of undefined tuples are those belonging to the union of the two models and not belonging to their intersection.

Example 9 Consider the integrated relation $Employee$ of Example 1. There are two alternative repairs which make the database consistent: $R_1 = \langle \{\}, \{Employee(Mary, 28, 20)\} \rangle$ which deletes from the database the tuple $Employee(Mary, 28, 20)$ and $R_1 = \langle \{\}, \{Employee(Mary, 31, 30)\} \rangle$ which deletes from the database the tuple $Employee(Mary, 31, 30)$. The query asking for name and age of every employee returns $\{(Peter, 47)\}$ as certain answer and $\{(Mary, 28), (Mary, 31)\}$ as uncertain answer. $\Box$

Although the computation of certain (and possible or uncertain) answers may be computationally expensive for general constraints, for simple classes of integrity constraints the computation can be very efficient.

Theorem 3 *Let D be a database, $\mathcal{FD}$ a set of functional dependencies over D, and Q a query. Then, $Q(D, \mathcal{FD})$ can be computed in polynomial time.* $\Box$

6 A SYSTEM PROTOTYPE

The framework for merging, repairing and querying inconsistent databases with functional and inclusion dependencies, presented in the previous sections has been implemented in a system prototype, developed at the University of Calabria. The overall architecture is reported in Figure 1. This system receives in input an Integration Technique $\mathcal{IT}$, consisting of a predefined operator or a set of SQL queries, a query $\mathcal{Q}$ and a repairing strategy $\mathcal{RS}$ (completing or cleaning) and outputs the answer *Ans*. The answers is partitioned into two parts: certain answers (set of tuples which are always true) and uncertain answers. The system is currently able to implement many of the integration operators, proposed in literature, such as the match join, the merging by majority, the merge and the prioritized merge operator. In more details, the integration methodology can be selected among the merging operators, defined in the literature, or most flexibly, the user can specify SQL queries, reflecting real user needs.

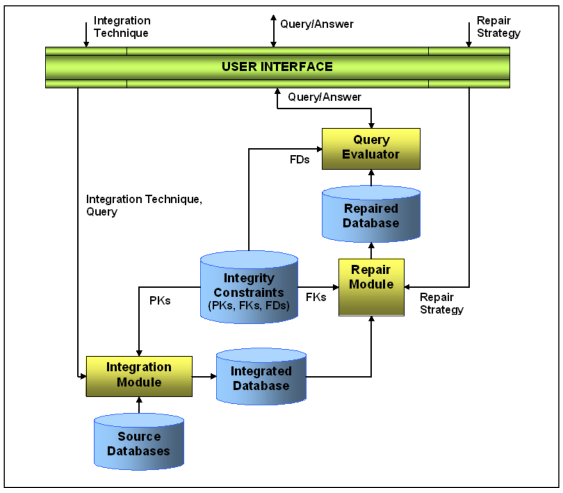

Figure 1: System architecture.

In the following we describe the main modules composing the system.

User Interface ($\mathcal{UI}$). This module receives an Integration technique $\mathcal{IT}$, (i.e. a predefined operator or a set of SQL queries), a query $\mathcal{Q}$ and a repair strategy $\mathcal{RS}$ (completing or cleaning) and outputs the answer *Ans*.

Integration Module ($\mathcal{IM}$). This module receives in input the Integration Technique or a set of SQL queries, and a (possible empty) query $\mathcal{Q}$ and constructs the corresponding integrated database ($\mathcal{IDB}$). In particular if both the query $\mathcal{Q}$ and the integration technique $\mathcal{IT}$ is provided the Integrated Database is constructed w.r.t. the set of relations directly or even indirectly involved in $\mathcal{Q}$. If just an integration technique $\mathcal{IT}$ is provided in input the Integrated Database is constructed w.r.t. the Source Databases.

Repairing Module ($\mathcal{RM}$). This module is responsible of solving foreign key inconsistencies using the strategy specified by the input parameter $\mathcal{RS}$ (cleaning or completing). In more details, it implements the repair strategy depicted in Section 4, by completing or cleaning the inconsistent database so that obtain a database consistent w.r.t the set of foreign keys.

The database obtained by cleaning or completing the $\mathcal{IDB}$ is called $\mathcal{CDB}$.

Query Evaluator Module ($\mathcal{QEM}$). This module receives the query $\mathcal{Q}$ and the set of functional dependencies and returns the answer, *Ans*, to the user interface. In particular, this module, implements the technique for managing inconsistent database, presented in Section 5, so that providing an answer evidencing the set of certain and uncertain tuples.

6.1 Implementing Merge Operators

In this section we show how to express integration operators. We consider two relations $S_1(K, A, B)$ and $S_2(K, A, C)$ where K, A, B and C denotes sets of attributes. In particular, $K = \{K_1, ..., K_m\} = pkey(S_1) = pkey(S_2)$ denotes the key of the two relations, $A = \{A_1, ..., A_n\} = attr(S_1) \cap attr(S_2) - K$ is the set of shared attributes not belonging to the key, $B = \{B_1, ..., B_p\} = attr(S_1) - attr(S_2)$ and $C = \{C_1, ..., C_q\} = attr(S_2) - attr(S_1)$ denote the set of attributes appearing in only one of the two relations. For the sake of brevity, we often use the set attributes K, A, B and C instead of the single attributes; for instance, we use $S_1.K = S_2.K$ to denote the equality condition $S_1.K_1 = S_2.K_1 \wedge \cdots \wedge S_1.K_m = S_2.K_m$. Moreover, in the following we will use use $\texttt{NULL}(B)$ to assign null values to the attributes in B and the standard operator $\texttt{COALESCE}(A_1, ..., A_n)$ to select the first not null value in the sequence.

The Match Join operator. The Match Join operator can be easily expressed by means of the following SQL statement:

```
CREATE VIEW $S(K, A, B, C)$ AS
   SELECT $K, A, B$, NULL($C$)    FROM $S_1$
   UNION
   SELECT $K, A$, NULL($B$), $C$    FROM $S_2$;

for each $X_i \in A \cup B \cup C$
   CREATE VIEW $T_i(K, A_i)$ AS
      SELECT $K, X_i$    FROM $S$
      WHERE $X_i$ IS NOT NULL

SELECT $T_1.K$,  $T_1.X_1$,  ...,  $T_m.X_{n+m+q}$
FROM $T_1$ OUTER JOIN $T_2$ ON $T_1.K = T_2.K \ldots$
    OUTER JOIN $T_n$ ON $T_{n-1}.K = T_n.K$;
```

The Merging by Majority Operator. The SQL statement expressing the Merging by Majority operator is the following:

```
CREATE VIEW $S(K, A, B, C)$ AS
   SELECT $K, A, B$, NULL($C$)    FROM $S_1$
   UNION
   SELECT $K, A$, NULL($B$), $C$    FROM $S_2$;

for each $X_i \in A \cup B \cup C$ {
   CREATE VIEW $W_i(K, X_i, Counter)$ AS
      SELECT $K, X_i$, COUNT($*$)    FROM $S$
      GROUP BY ($K, X_i$);

   CREATE VIEW $T_i(K, X_i)$ AS
      SELECT $K, X_i$    FROM $W_i$
      WHERE NOT EXISTS(
         SELECT *    FROM $W_i$ AS $W_i'$
         WHERE $W_i.K = W_i'.K$ AND
               $W_i.Counter < W_i'.Counter$); }

SELECT $T_1.K$,  $T_1.X_1$,  ...,  $T_{n+p+q}.X_{n+p+q}$
FROM $T_1, ..., T_{n+p+q}$
WHERE $\mathcal{T}_1.K = \mathcal{T}_2.K$   AND ...
       AND $\mathcal{T}_{n+p+q-1}.K = \mathcal{T}_{n+p+q}.K$;
```

The Merge operator. The Merge operator can be easily expressed by means of the following SQL statement:

```
SELECT $S_1.K$,  $S_1.B$,  COALESCE($S_1.A_1, S_2.A_1$),  ...,
          COALESCE($S_1.A_n, S_2.A_n$), $S_2.C$
FROM $S_1$ LEFT OUTER JOIN $S_2$ ON $S_1.K = S_2.K$
UNION
```

SciTePress Science and Technology Publications

```
SELECT  $S_2.K, S_1.B,$ COALESCE($S_2.A_1, S_1.A_1$), ..,
            COALESCE($S_2.A_n, S_1.A_n$), $S_2.C$
FROM $S_1$ LEFT OUTER JOIN $S_2$ ON $S_1.K = S_2.K$
```

The Prioritized Merge operator. The Prioritized Merge operation $S_1 \triangleleft S_2$, can be easily expressed by means of an SQL statement, as follows:

```
SELECT  $S_1.K, S_1.B,$ COALESCE($S_1.A_1, S_2.A_1$), ..,
            COALESCE($S_1.A_n, S_2.A_n$), $S_2.C$
FROM $S_1$ LEFT OUTER JOIN $S_2$ ON $S_1.K = S_2.K$
UNION
SELECT  $S_2.K,$ NULL($B$), $S_2.A, S_2.C$
FROM    $S_2, S_1$
WHERE  $S_2.K$ NOT IN (SELECT $S_1.K$ FROM $S_1$)
```

6.2 Computing Consistent Answer

Given a query $\mathcal{Q}$ the computation of the consistent answer, $Consistent\_Ans$, is performed as follows: i) firstly, the relation Ans, obtained by evaluating the query $\mathcal{Q}$ over the (partial) repaired database and containing both "certain" and "undefined" tuples is constructed; ii) secondly, the relation $Inconsistent\_Ans$ containing the tuples of Ans which do not satisfy some of the functional dependency involving attributes in $\mathcal{Q}$, is constructed; iii) finally, the $Consistent\_Ans$, is carried out by selecting tuples of Ans which are not in $Inconsistent\_Ans$. Consider, for instance, the Example 1 with the functional dependency $Name \rightarrow Age, Salary$ and suppose to perform a query asking for *the name and the age of Employees*. The relation Ans is obtained by trivially selecting Name and Age from Employee, whereas the relation $Inconsistent\_Ans$ can be obtained by means of the following SQL view:

```
CREATE VIEW Inconsistent_Ans AS
SELECT Ans.* FROM  Ans, Employee E
WHERE  Ans.Name = E.Name
AND  Ans.Age <> E.Age
```

The consistent answer, $Consistent\_R$, can be obtained by means of the following SQL view:

```
CREATE VIEW Consistent_Ans AS
SELECT * FROM  Ans
EXCEPT
SELECT * FROM  Inconsistent_Ans
```

REFERENCES

Abiteboul, S., Hull, R., Vianu, V. *Foundations of Databases*. Addison-Wesley, 1994.

Agarwal, S., Keller, A. M., Wiederhold, G., Saraswat, K., Flexible Relation: an Approach for Integrating Data from Multiple, Possibly Inconsistent Databases. *ICDE*, 1995.

Arenas, M., Bertossi, L., Chomicki, J., Consistent Query Answers in Inconsistent Databases. *Proc. PODS 1999*, pp. 68–79, 1999.

Baral, C., Kraus, S., Minker, J., Combining Multiple Knowledge Bases. *IEEE-TKDE*, 3(2): 208-220 (1991)

Breitbart, Y., Multidatabase interoperability. *Sigmod Record 19(3)* (1990), 53–60.

Bry, F., Query Answering in Information System with Integrity Constraints,*IICIS*, pp. 113-130, 1997.

Cali, A., Calvanese, D., De Giacomo, G., Lenzerini, M., Data Integration under Integrity Constraints. *CAiSE*, pp. 262-279, 2002.

Dung, P. M. ,Integrating Data from Possibly Inconsistent Databases. *COOPIS*, pp. 58-65, 1996.

Grant, J., Subrahmanian, V. S., Reasoning in Inconsistent Knowledge Bases. *IEEE-TKDE*, 7(1): 177-189, 1995.

Greco, S., Zumpano, E., Querying Inconsistent Database *LPAR*, pp. 308-325, 2000.

Greco, G., Greco, S., Zumpano, E., A Logic Programming Approach to the Integration, Repairing and Querying of Inconsistent Databases. *ICLP* pp. 348-364, 2001.

Greco, S., Pontieri, L., Zumpano, E., Integrating and Managing Conflicting Data. *Ershov Memorial Conference* pp. 349-362, 2001.

Levy, A., Rajaraman, A., Ordille, J., Querying heterogeneous nformation sources using source descriptions. *VLDB*, pp. 251–262, 1996.

Lin, J., Mendelzon, A. O., Knowledge Base Merging by Majority, in R. Pareschi and B. Fronhoefer (eds.), *Dynamic Worlds*, Kluwer, 1999. Kluwer, 1999.

Lin, J., A Semantics for Reasoning Consistently in the Presence of Inconsistency. *AI*, 86(1), pp. 75-95, 1996.

Lin, J., Integration of Weighted Knowledge Bases. *Artificial Intelligence*, Vol. 83, No. 2, pages 363-378, 1996.

Pradhan, S., J. Minker, J., Subrahmanian, V.S., Combining Databases with Prioritized Information *JIIS*, 4(3), pp. 231-260, 1995.

Subrahmanian, V. S., Amalgamating Knowledge Bases. *ACM-TODS*, Vol. 19, No. 2, pp. 291-331, 1994.

Yan, L.L., Ozsu, M. T., Conflict Tolerant Queries in Aurora *Coopis*, pp. 279-290, 1999.

Ullman, J. D., *Principles of Database and Knowledge-Base Systems*, Vol. 1, Computer Science Pressingness, 1998.

Wiederhold, G., Mediators in the architecture of future information systems. *IEEE Computer* 25(3): 38–49, 1992.